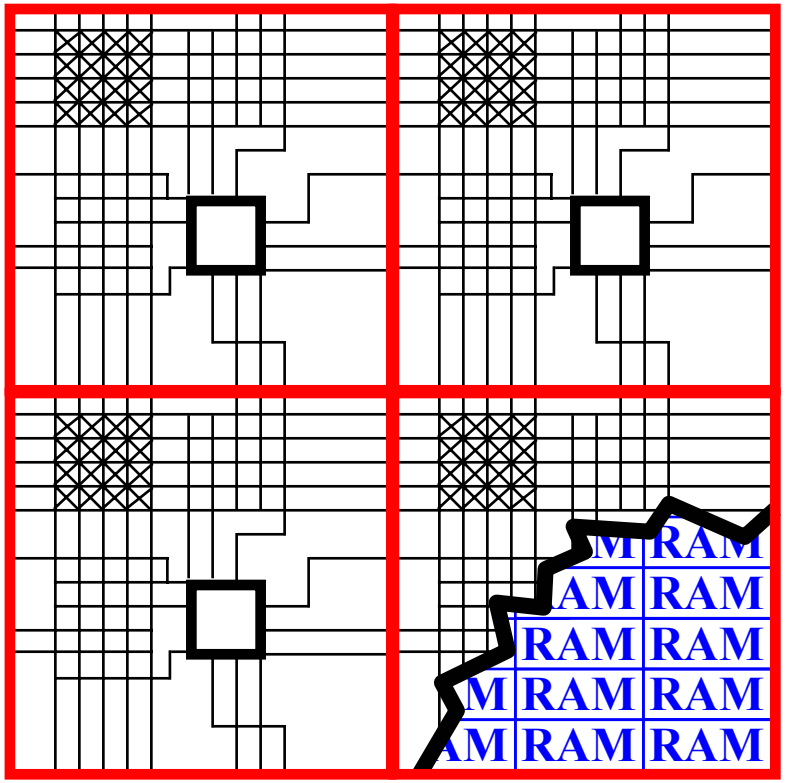
Variable Precision Analysis for FPGA Synthesis

Mark L. Chang, Scott Hauck

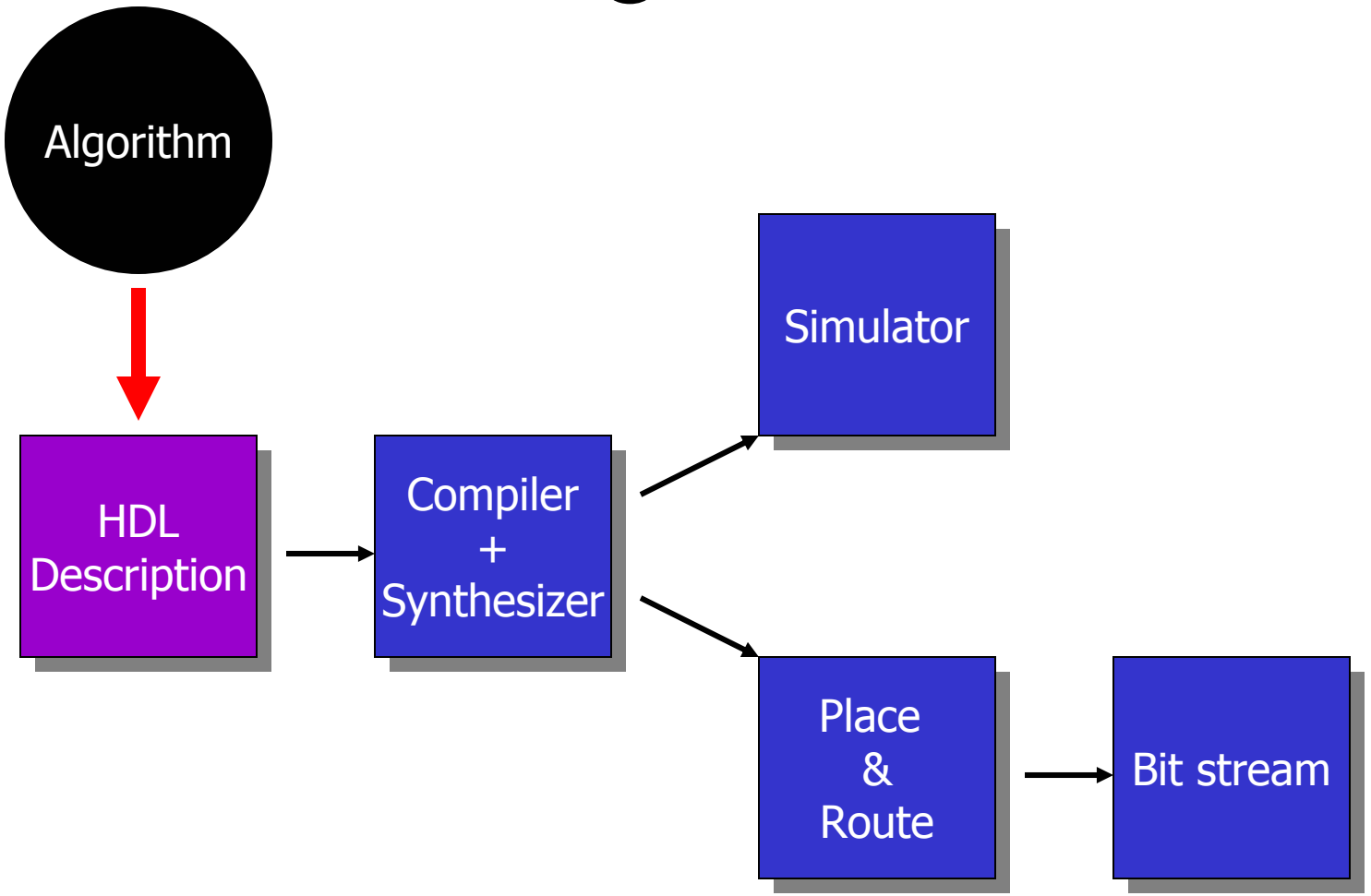
University of Washington

Field Programmable Gate Arrays

- Programmable logic blocks and interconnection
- Infinitely and quickly reconfigurable
- Can be orders of magnitude faster than general-purpose processors



Design Flow


Algorithm
HDL Description
Compiler + Synthesizer
Simulator
Place & Route
Bit stream

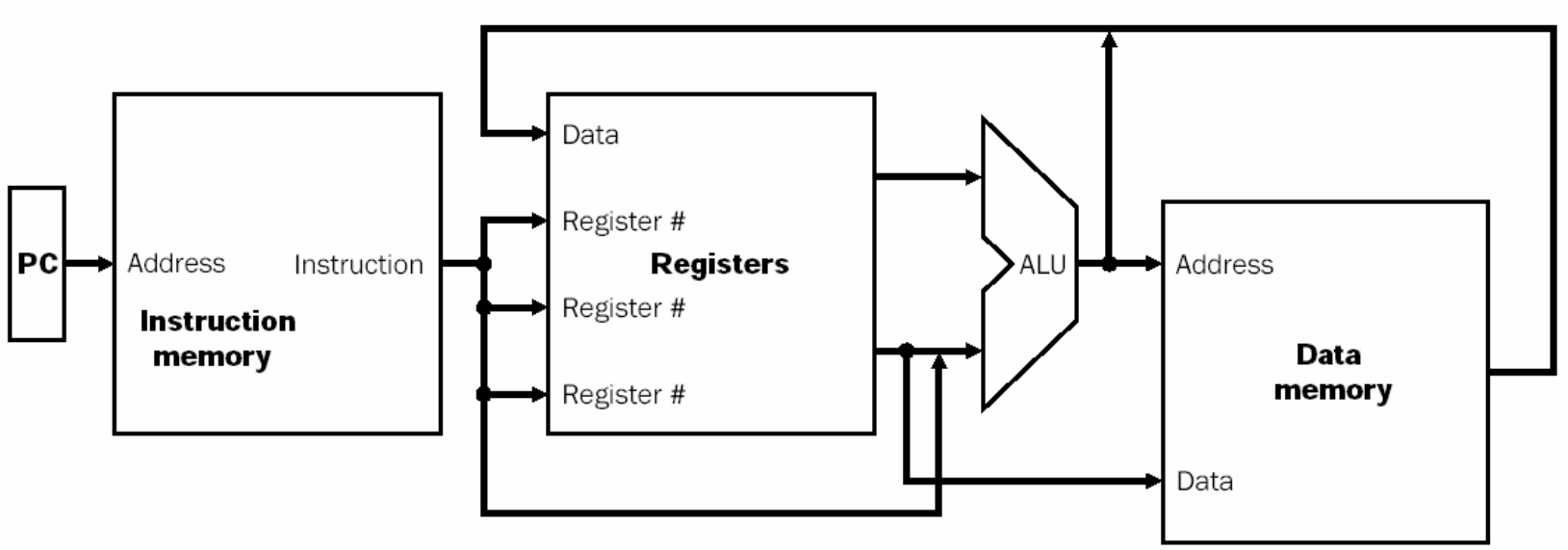
Why Precision Analysis?

- Algorithm originally written for general-purpose processors
- Want to implement on an FPGA

- Two very different computing paradigms

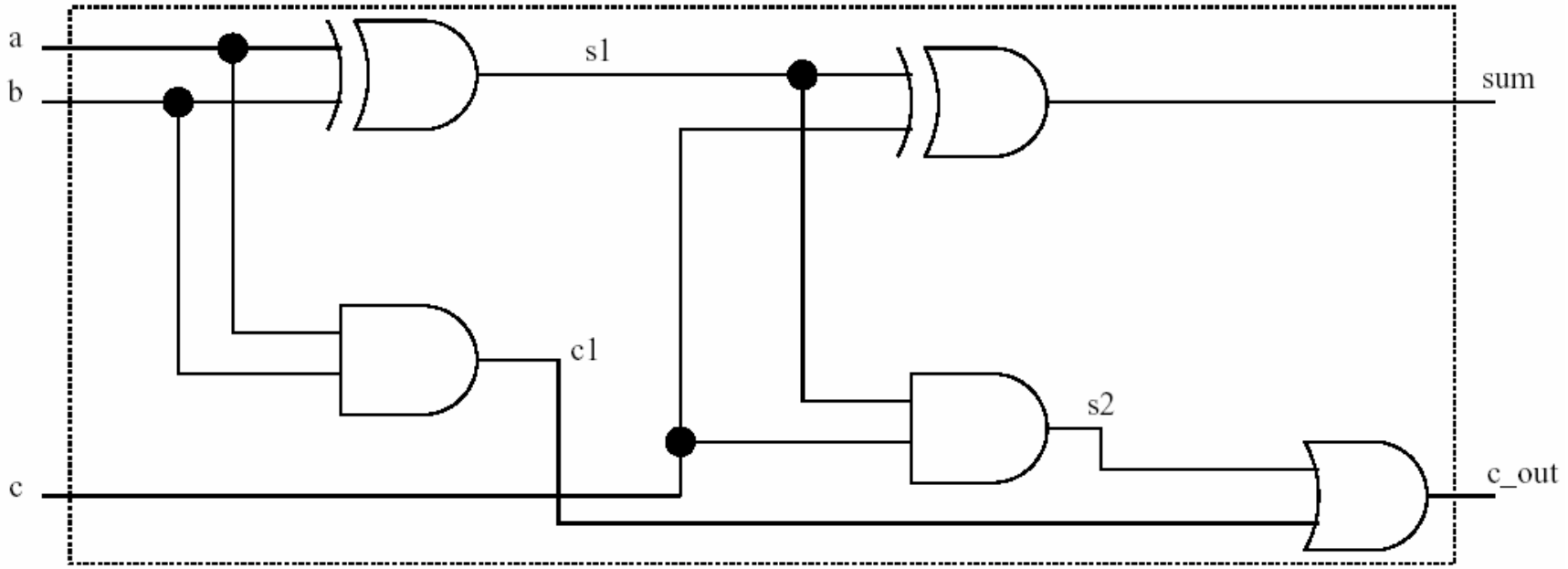
General-Purpose Processor

- Word-level device
- Data type abstractions
  - `char, int, double`



FPGA

- Bit-level device
- Bit-exact representation in circuit



Conversion is Difficult

- Entire width of datapath not necessarily used
- Need to match datapath size to algorithm
- Maintain correctness
- Achieve optimality
- Software designers rarely think about overflow and underflow

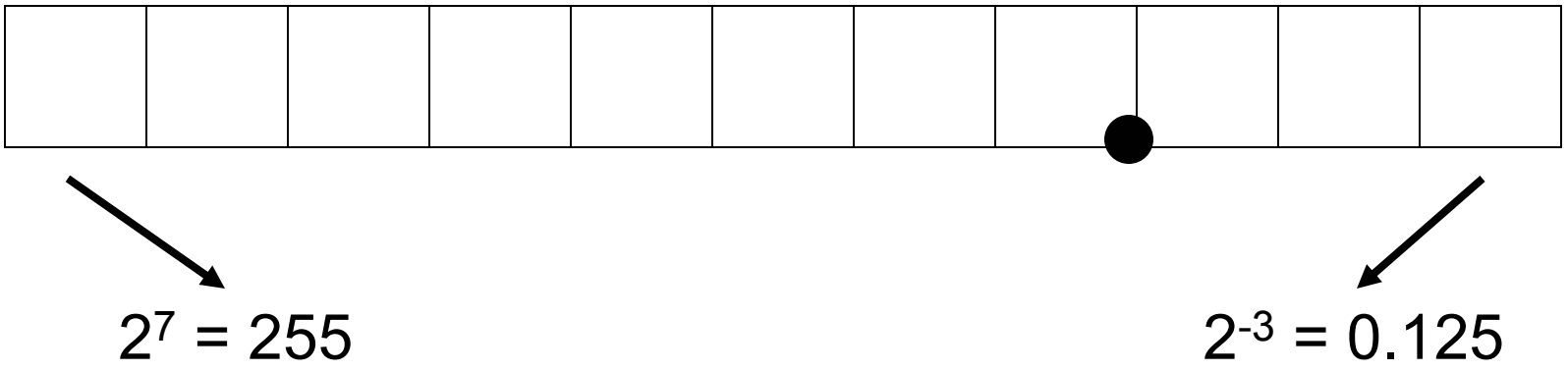
Manual Precision Analysis

- What are the provable precision requirements of my algorithm?
- What are the actual precision requirements of my data sets?
- What are the effects of fixed-precision on my results?
- Where should I optimize?

Précis: A Design-Time Tool

- Provable precision requirements:
  Propagation engine
- Actual precision requirements:
  Range finding
- Effects of fixed-precision:
  Simulation
- Where should I optimize:
  Slack analysis

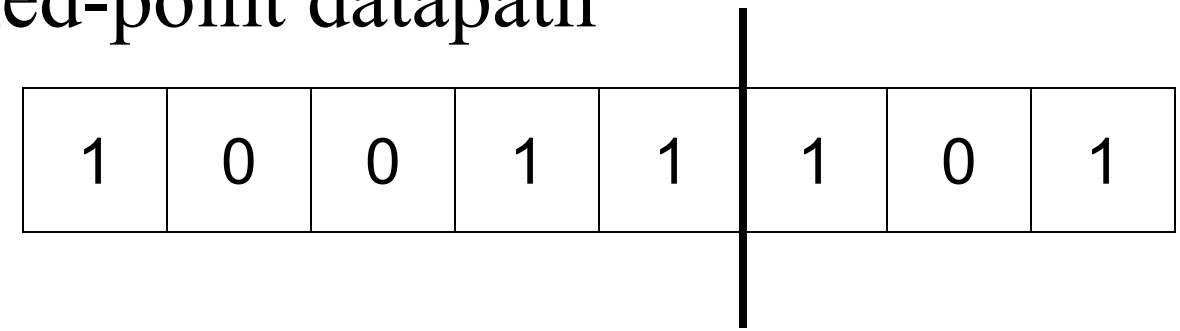
The Least Significant Bit



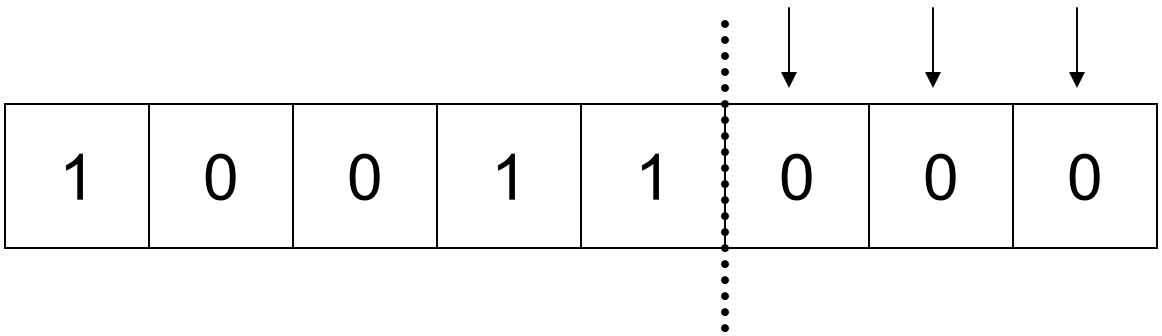
- Maximum range = 255
- Quantized into multiples of 0.125
- Need methodology to determine the “best” LSB position

Methodology

- Typically, truncation is used to model a fixed-point datapath



- Instead, replace with known constants

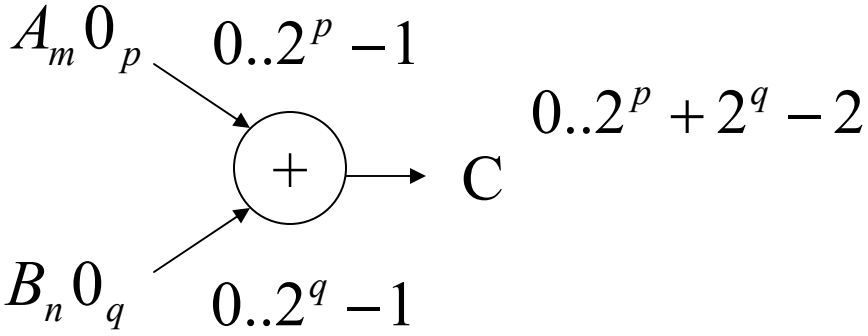


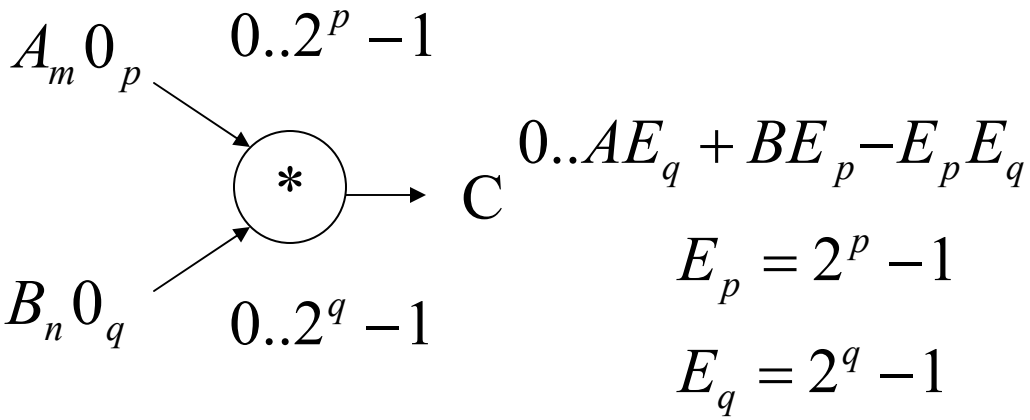
Methodology

- Reducing input bit widths to an operator reduces the area requirement
- This same relationship holds if we substitute zeros instead of truncate
- Use wires to represent substituted zeros
- Almost free in terms of area

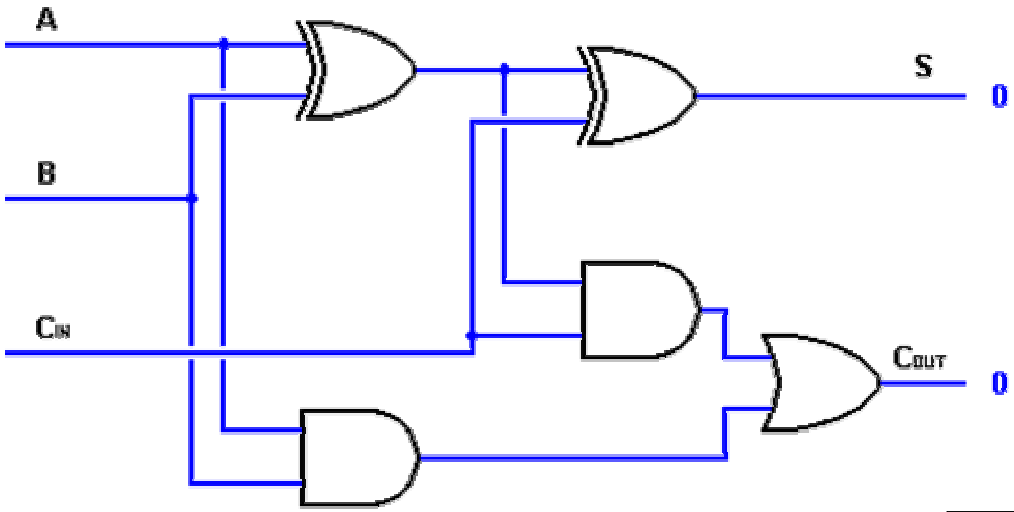
Error Models

- Zero substitution introduces error



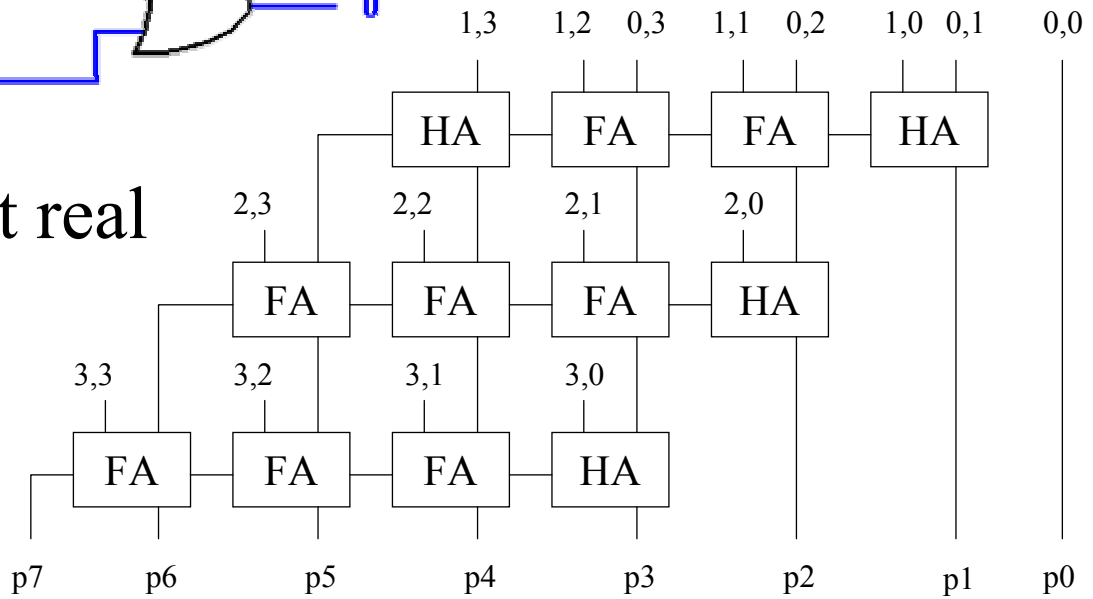


Area Models

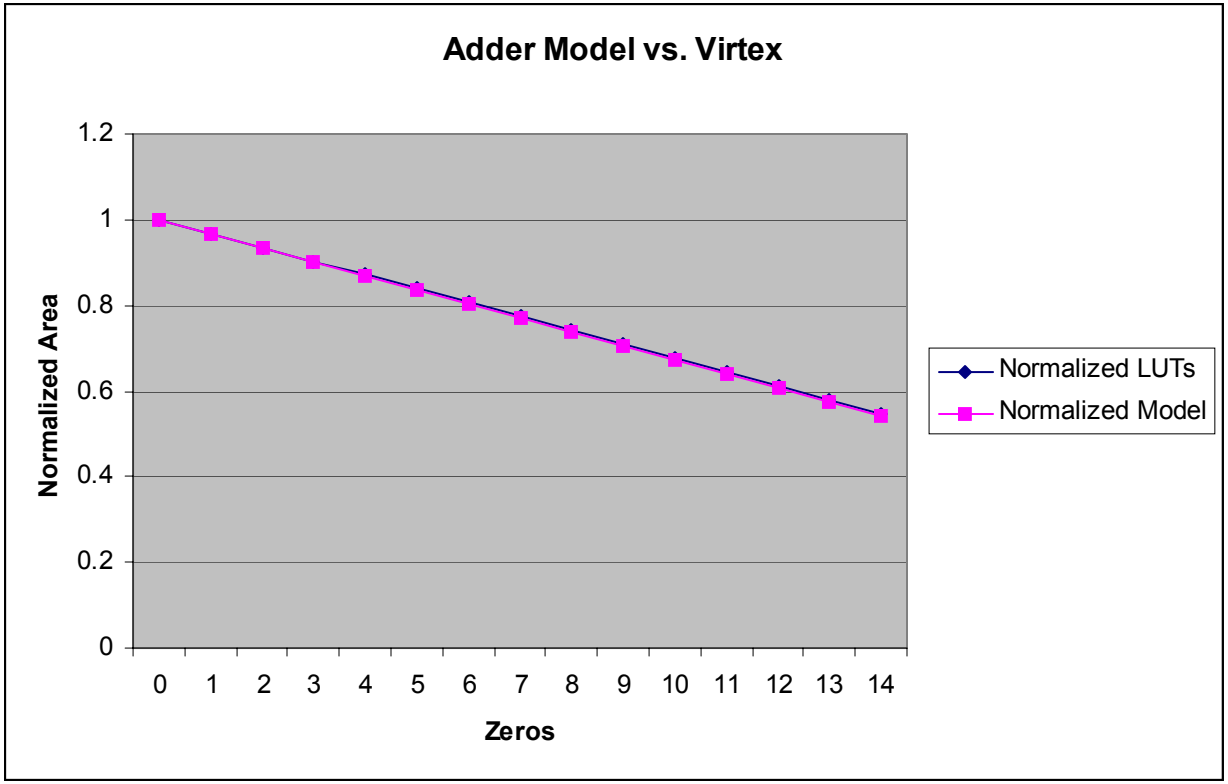


- Utilize 2-LUT model

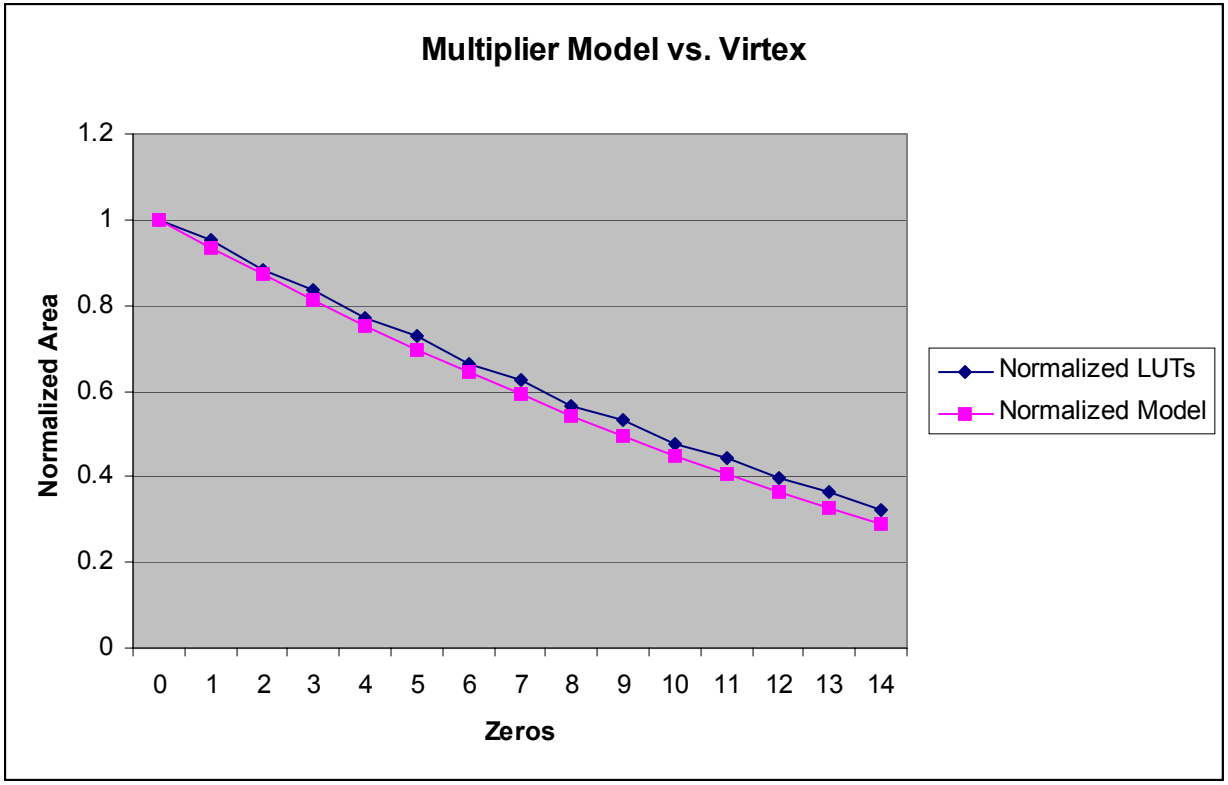
- Compare against real hardware implementation



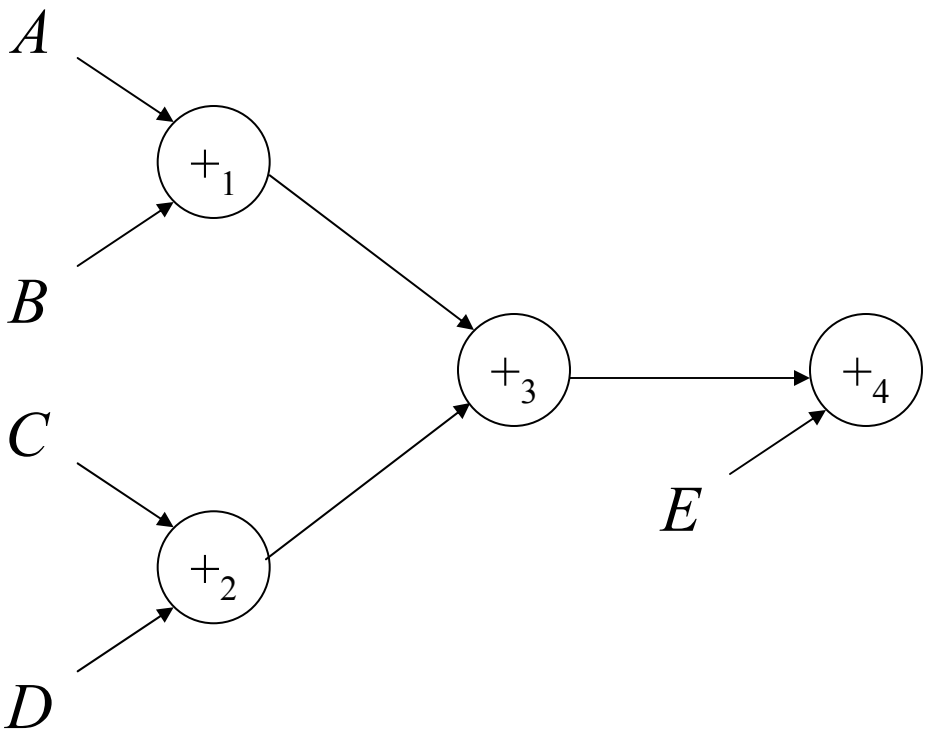
Adder Area Model



Multiplier Area Model



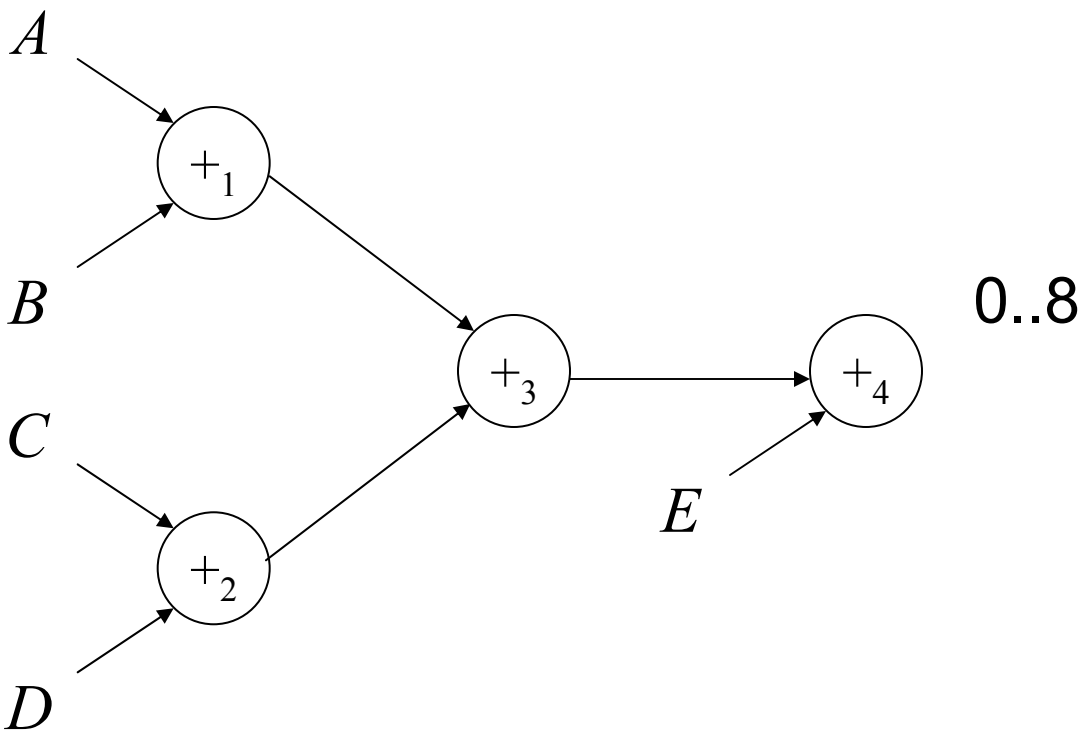
Steering Error



Assume all inputs are 4-bit

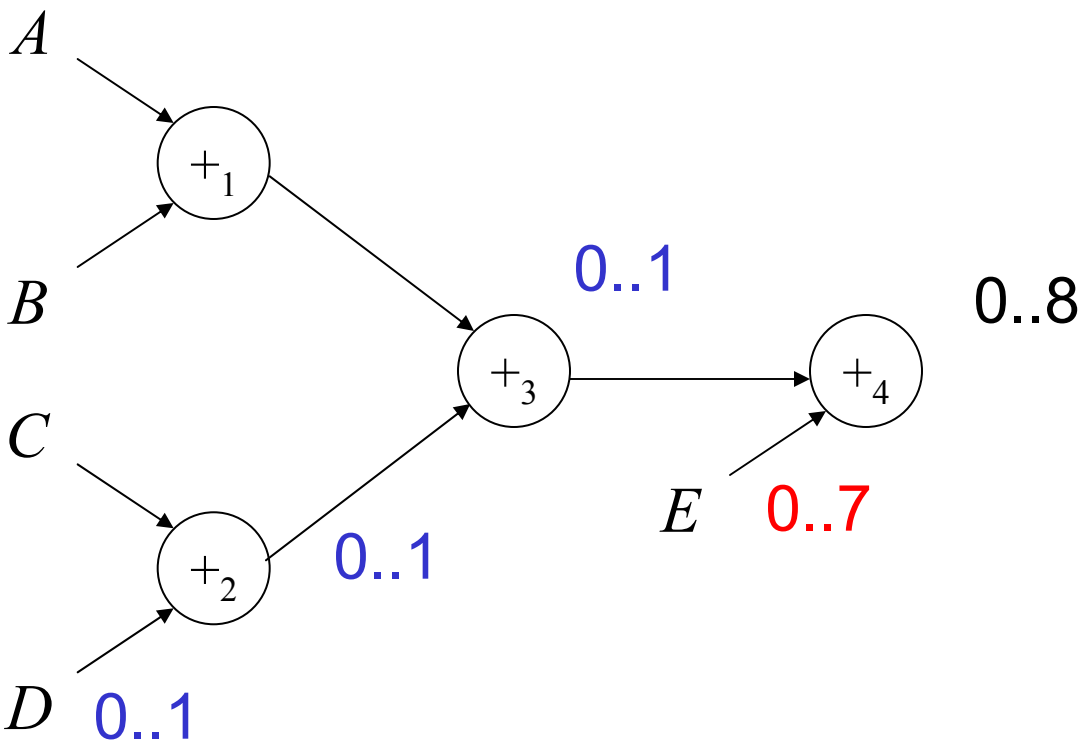
77 2-LUTs for base implementation

Steering Error



Tolerate 0..8 error on output

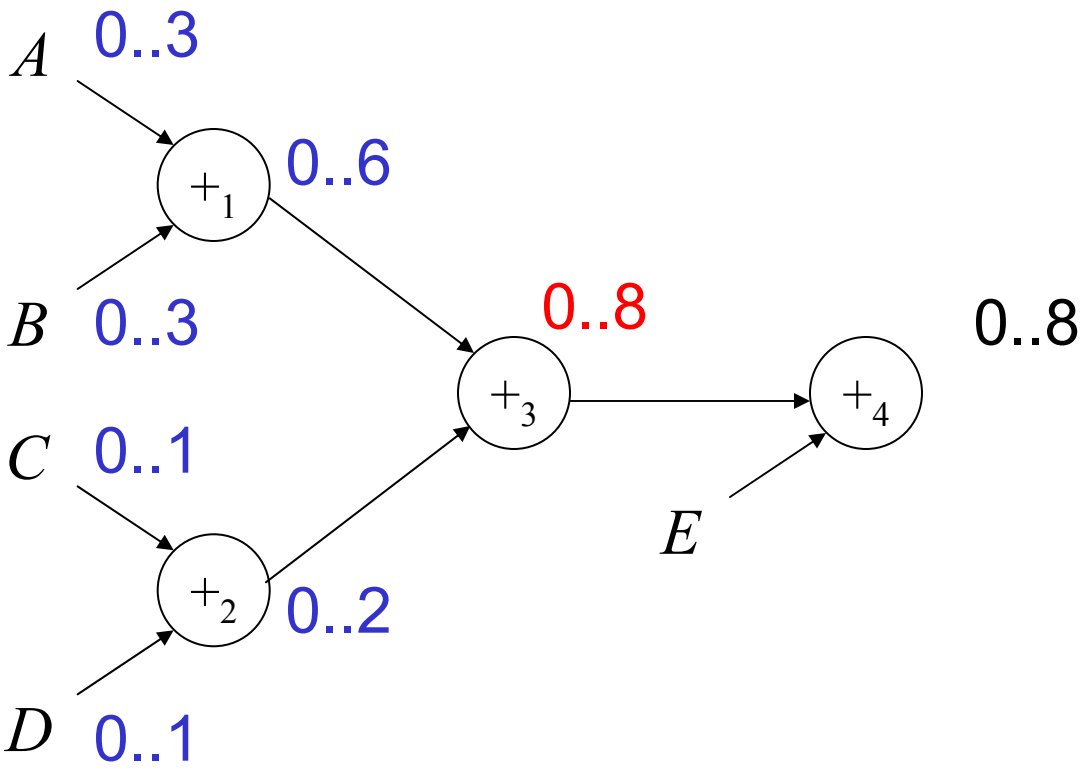
Steering Error



Steer toward shorter branch

57 2-LUTs

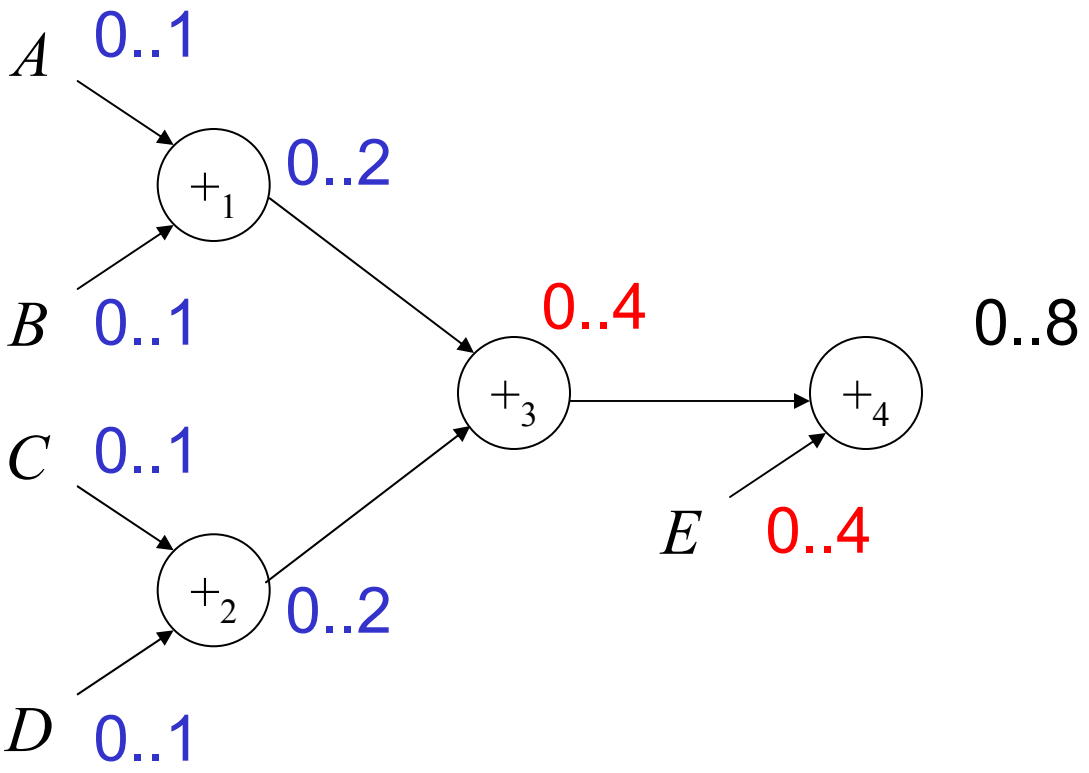
Steering Error



Steer toward longer branch

47 2-LUTs

Steering Error



Balanced error

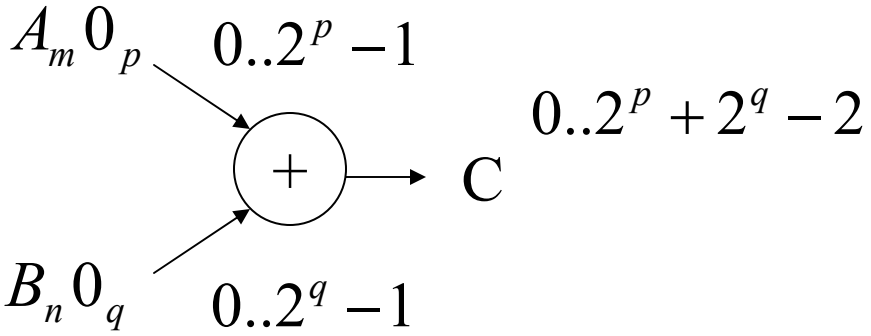
52 2-LUTs

Guidelines for Precision Optimization

- Area benefits are quantized to the number of whole-bit substitutions that can be made
  - An error range of 0..2 can not be fully utilized at a single input
  - An error range of 0..2 can be utilized if split between two inputs
  - If error cannot be distributed completely, overall output error can be reduced (take up slack)
- Area benefits increase as error is steered toward longer branches
  - More opportunity to reduce the area of ancestor nodes

Rethinking Error

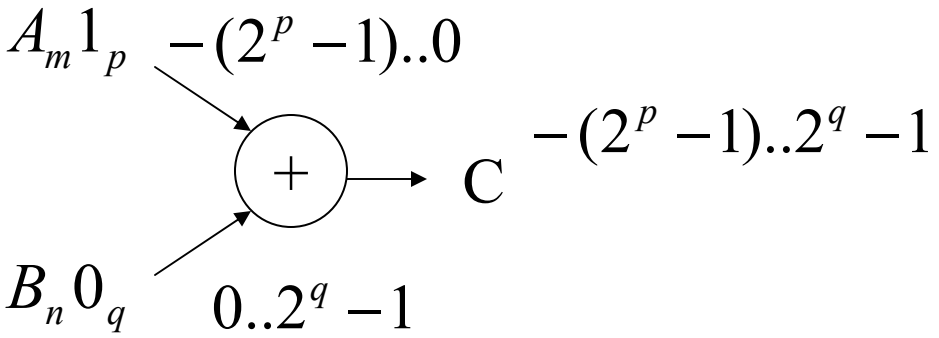
- With straight substitution of zeros, the error range is entirely positive

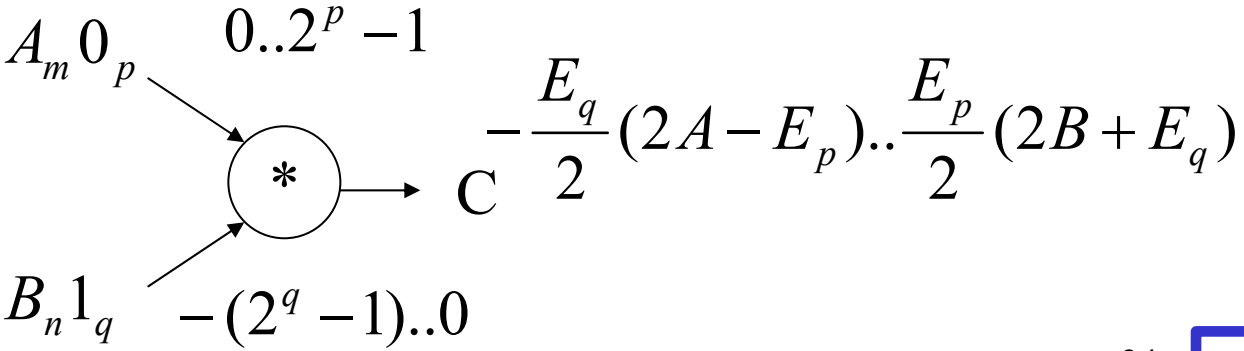


- Reconsider the error as the net distance from the real value
- Achieves twice the error range while maintaining the same relative level of accuracy

Renormalization

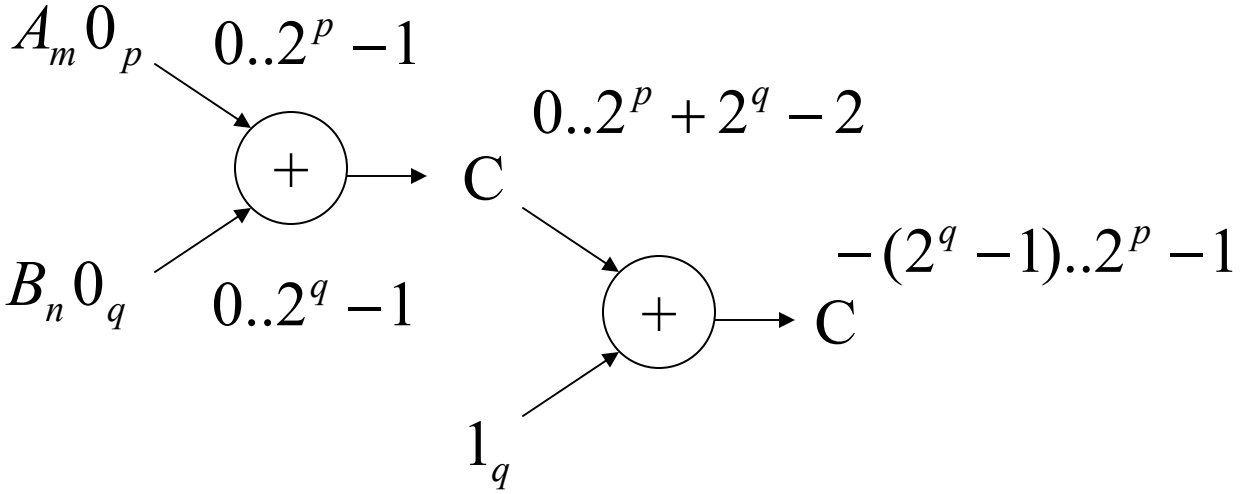
- Can bias the output error range by substituting ones and zeros strategically





Renormalization

- Alternatively, renormalize later by adding a constant to bias the error range



Conclusions

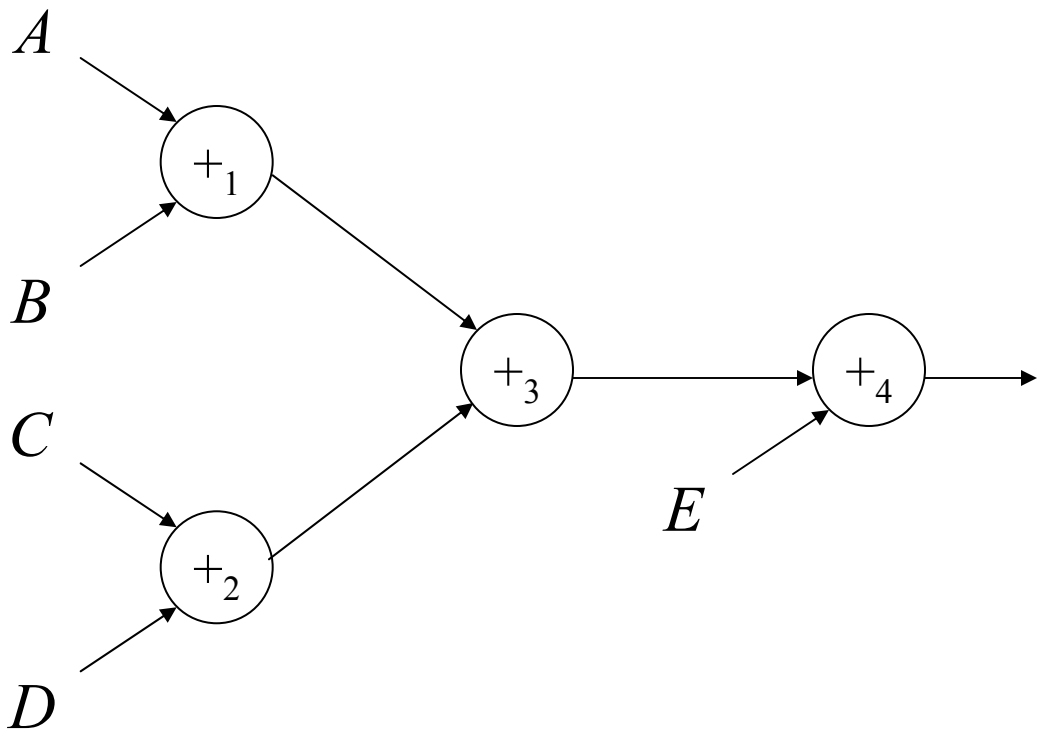
- Precision optimization of circuit datapath is a major implementation detail
- Poor support from a tools perspective
- Proposed methodologies for precision optimization of least significant bit position
  - Area and Error models
  - Optimization guidelines
  - Renormalization alternatives

Future Directions

Active vs. Lazy Renormalization

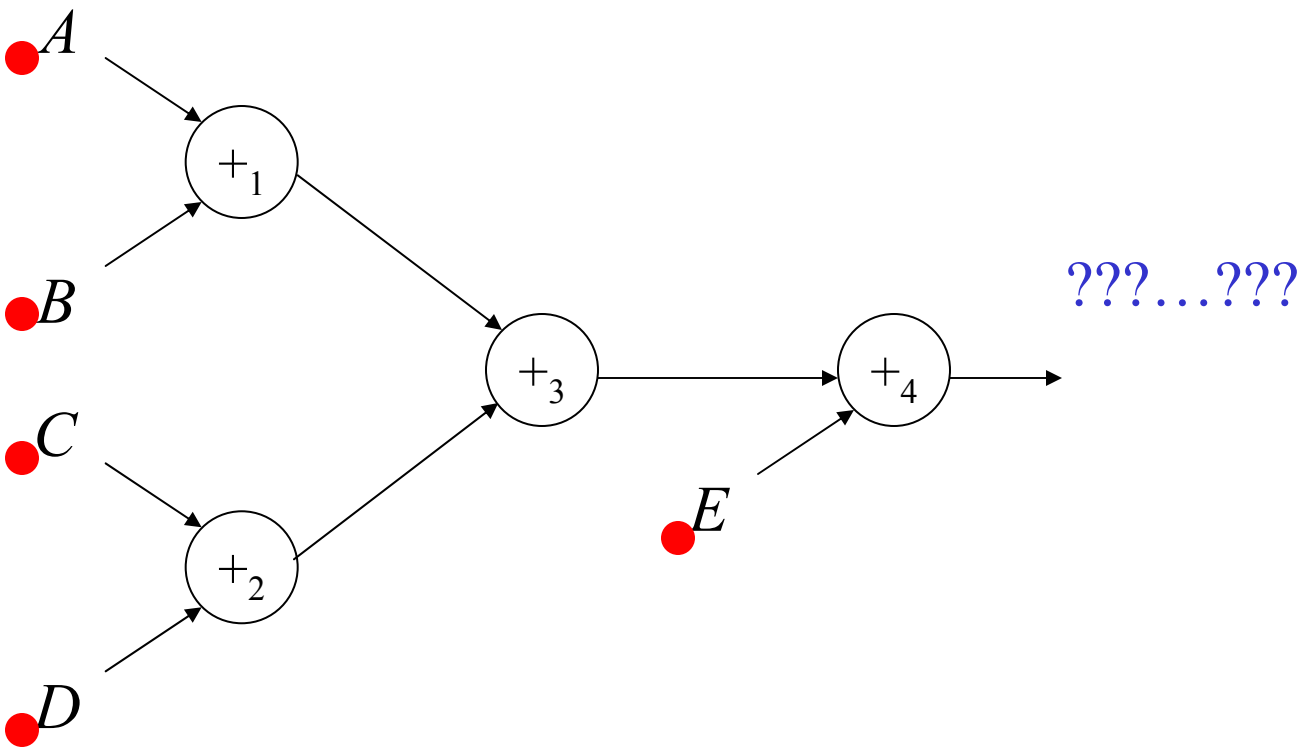
Active vs. Lazy Renormalization

- When/How should we perform renormalization?



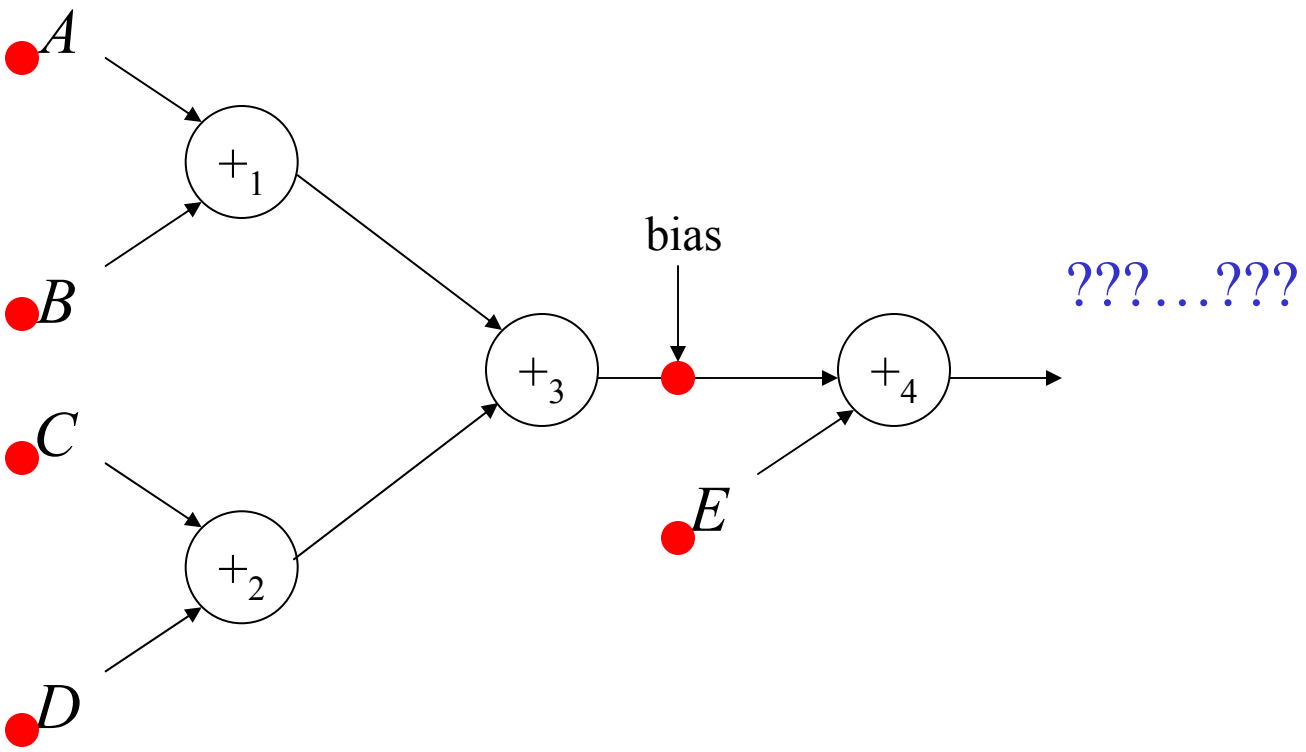
Active vs. Lazy Renormalization

- Do the primary inputs give us enough control?



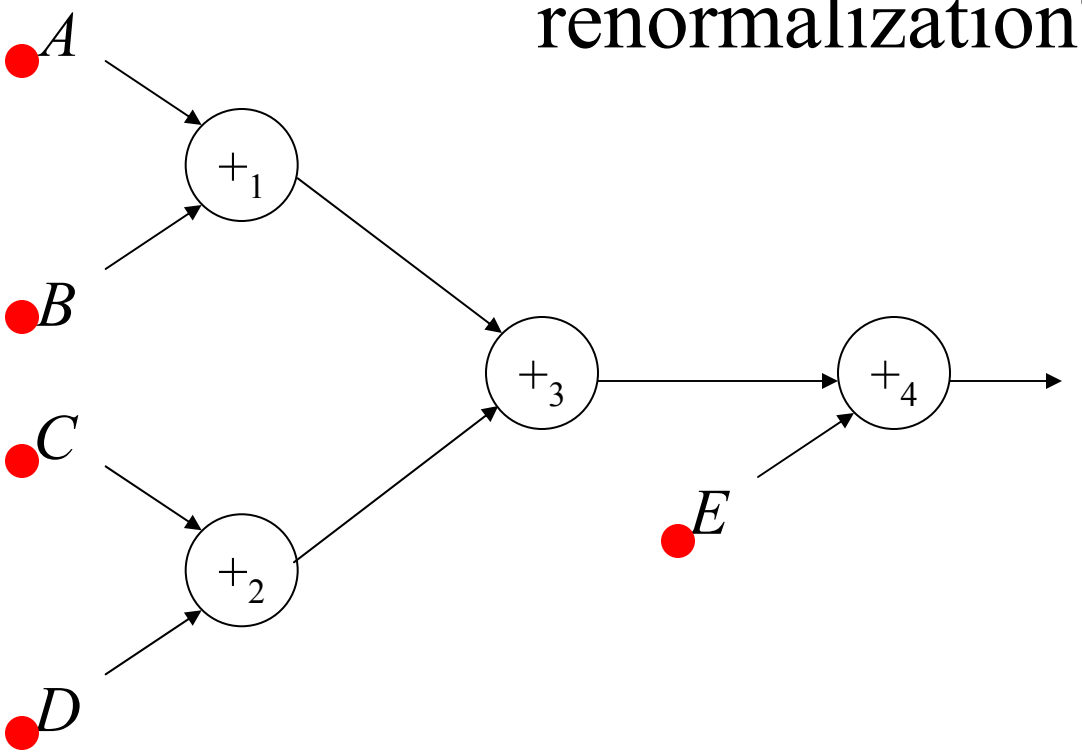
Active vs. Lazy Renormalization

- When do we need to insert bias points?



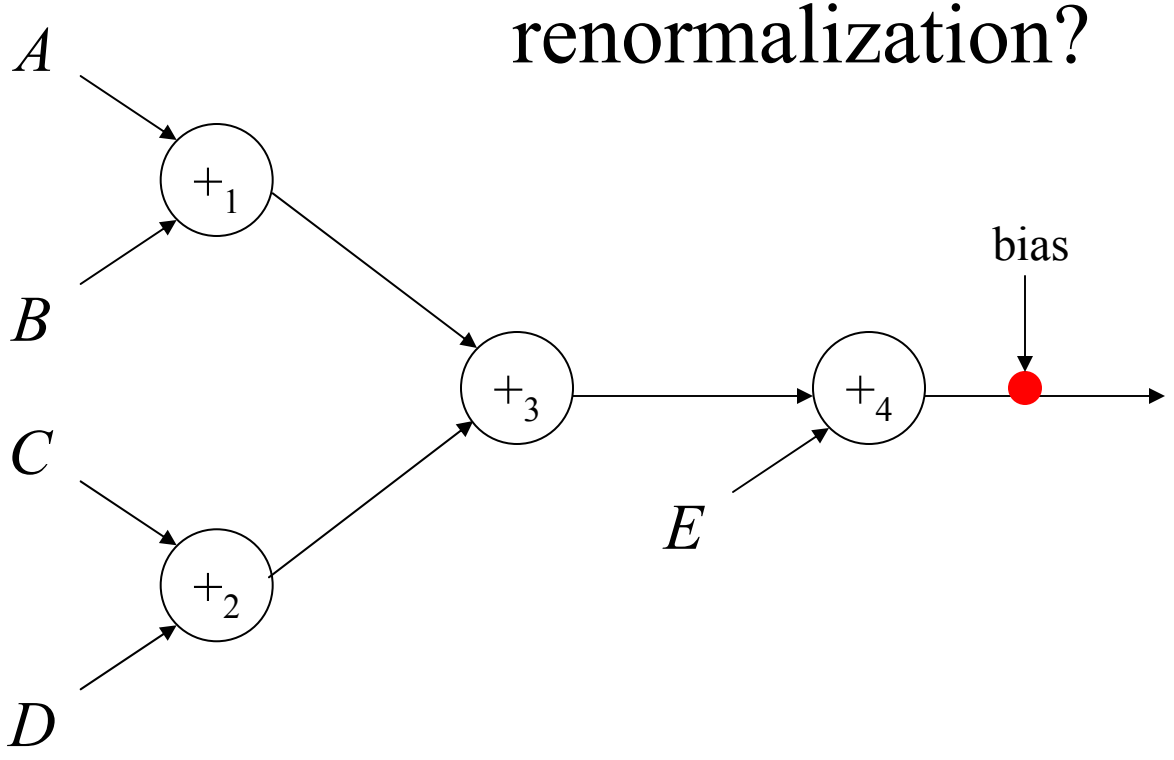
Active vs. Lazy Renormalization

- What is the area impact of various types of renormalization?



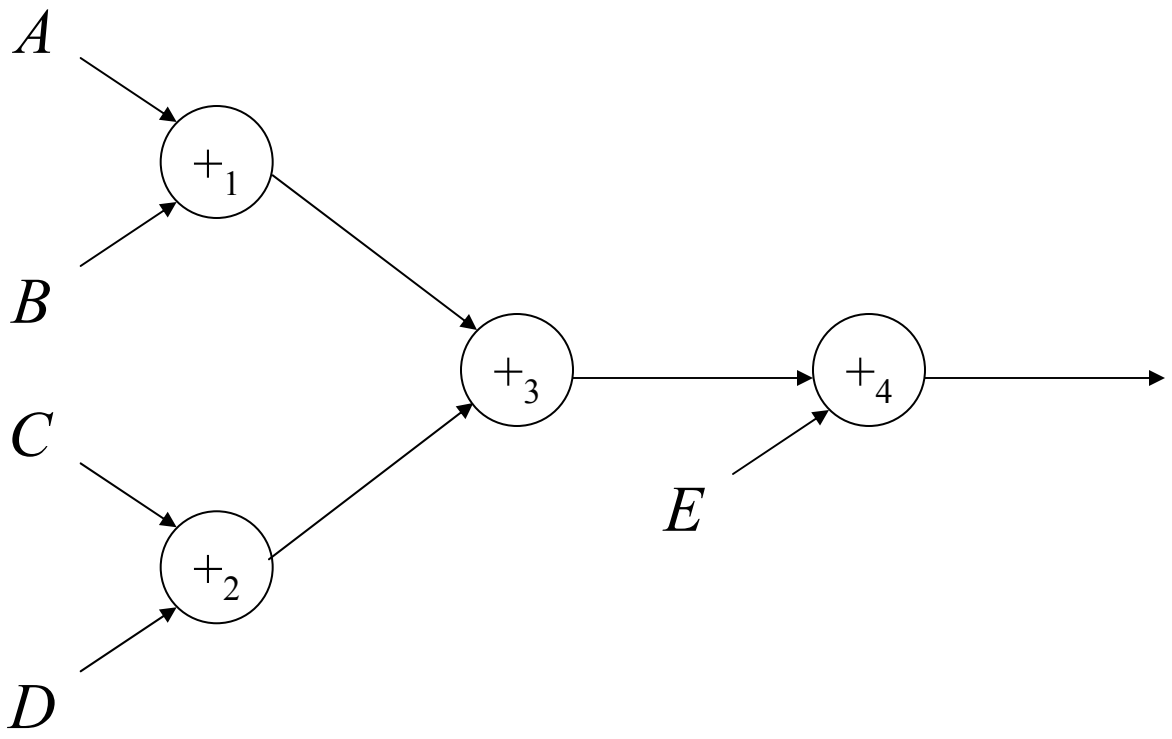
Active vs. Lazy Renormalization

- What is the area impact of various types of renormalization?



Active vs. Lazy Renormalization

- Is there a "best" way to renormalize?



Renormalization Approaches

- Determine area effects of renormalization
  - Single operator node (active)
  - Constant bias (lazy)
- Determine area effects these renormalization methods have on child nodes
- With area and error, we can deduce an algorithm for type and placement